Blue Gene/L advanced diagnostics environment

M. E. Giampapa
R. Bellofatto
M. A. Blumrich
D. Chen
M. B. Dombrowa
A. Gara
R. A. Haring
P. Heidelberger
D. Hoenicke
G. V. Kopcsay
B. J. Nathanson
B. D. Steinmacher-Burow
M. Ohmacht
V. Salapura
P. Vranas

This paper describes the Blue Gene®/L advanced diagnostics environment (ADE) used throughout all aspects of the Blue Gene/L project, including design, logic verification, bring-up, diagnostics, and manufacturing test. The Blue Gene/L ADE consists of a lightweight multithreaded coherence-managed kernel, runtime libraries, device drivers, system programming interfaces, compilers, and host-based development tools. It provides complete and flexible access to all features of the Blue Gene/L hardware. Prior to the existence of hardware, ADE was used on Very high-speed integrated circuit Hardware Description Language (VHDL) models, not only for logic verification, but also for performance measurements, code-path analysis, and evaluation of architectural tradeoffs. During early hardware bring-up, the ability to run in a cycle-reproducible manner on both hardware and VHDL proved invaluable in fault isolation and analysis. However, ADE is also capable of supporting high-performance applications and parallel test cases, thereby permitting us to stress the hardware to the limits of its capabilities. This paper also provides insights into system-level and device-level programming of Blue Gene/L to assist developers of high-performance applications to more fully exploit the performance of the machine.

Introduction

The Blue Gene*/L (BG/L) advanced diagnostics environment (ADE) has been used during all aspects of the BG/L project, including design, logic verification, bring-up, manufacturing test, and system diagnostics [1]. ADE provides a scalable productive programming environment that ranges from a single-node simulation at a few cycles per second up to high-performance parallel partitions comprising tens of thousands of nodes. The availability of ADE in the early design phases enabled feedback into the architecture in time to evaluate design tradeoffs and made it possible to perform code-path analysis and to verify device programming interfaces. Architectural studies using ADE include single-node memory system and computational kernel performance measurements. In the scaling phase of the BG/L project, ADE has been used for studies of network performance, scalability, and machine reliability. Test cases built atop ADE have been organized into the regression suite used in verification [2] and the manufacturing test and diagnostics suites used in production.

ADE host-based software consists of compilers, development tools, configuration and personalization tools, and a partition-management console. ADE node software provides complete and flexible access to all features of the BG/L compute (BLC) chip hardware, shown in **Figure 1**; it consists of a lightweight multithreaded coherence-managed kernel, runtime libraries, device drivers, and system programming interfaces. The design and modularity of the ADE kernel closely follow the BLC node architecture. The BLC chip contains two processor complexes, each consisting of an embedded 32-bit IBM PowerPC* 440 (PPC440) [3] processor and a custom double floating-point unit (FPU) [4] built from two 64-bit Book E [5] FPUs. Architecturally, these two processor complexes are identical, with full access to all on-chip facilities managed by the ADE kernel and system programming interfaces.

0018-8646/05/$5.00 © 2005 IBM

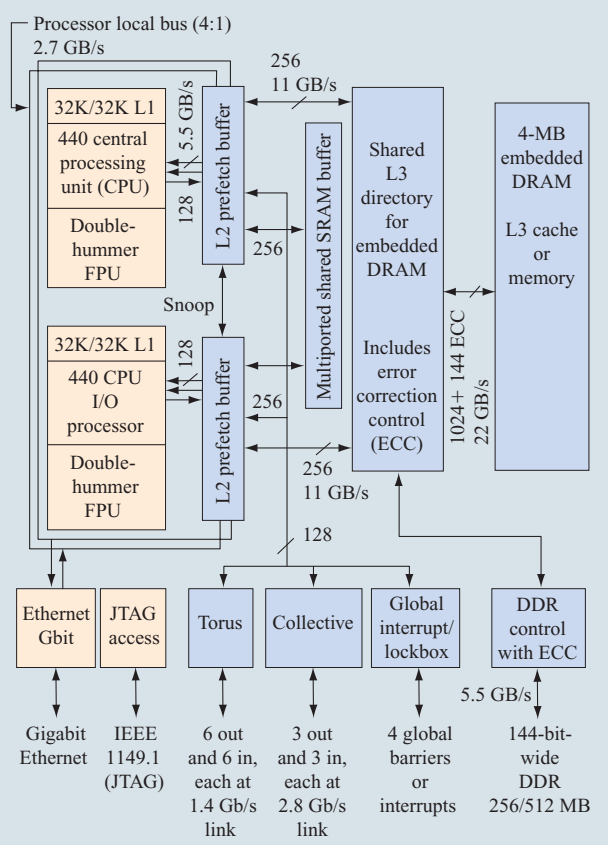

Figure 1

Blue Gene/L compute (BLC) chip architecture. Yellow shading indicates off-the-shelf cores. ©2002 IEEE. Reprinted with permission from G. Almasi et al., "Cellular Supercomputing with System-on-a-Chip," *Digest of Technical Papers*, 2002 IEEE International Solid-State Circuits Conference.

The BLC memory system interface contains three levels of on-chip cache.

Each core contains separate 32-KB level 1 (L1) instruction and data caches. Hardware does not maintain coherence at L1. Although this burdens software with coherence management where necessary, it provides an overall memory system performance gain by eliminating hardware snoop overheads and bus coherence protocol traffic between the processors.

The L2 and L3 caches are weakly ordered coherent. The L2 cache for each processor complex provides sequential prefetching over multiple datastreams and supports coherence via snooping between the L2 caches. L3 uses 4 MB of embedded dynamic random access memory (DRAM), which is partitionable between cache and directly addressed memory. L3 interfaces with a double-data-rate (DDR) controller that supports 256 MB to 2 GB of external DDR. A 16-KB static random access memory (SRAM) is used for booting, communication with the host service processor, and shared high-speed storage.

Communication services provided by the ADE kernel and system programming interfaces support the five BG/L networks. The BLC chip contains three high-performance networks: a three-dimensional torus network [6], a collective network,[1] and a global interrupt network. There are two external network interfaces: a Gigabit Ethernet used on input/output (I/O) nodes and a JTAG network (IEEE Standard 1149.1) that interfaces all nodes with the external control and monitoring system. Closely associated with the JTAG network, the BLC chip contains a test interface unit used by the ADE kernel to control unit resets, clock enables, and low-level hardware debug, and to discover configuration information set up by the service processor control system, described in [7].

Not shown in Figure 1 are three devices that span all units and are critical to kernel and diagnostics software:

- *BG/L interrupt controller (BIC):* The BIC gathers and prioritizes interrupt signals from all devices for presentation to the kernel as low-priority standard interrupts, high-priority critical interrupts, or urgent machine checks.
- *Universal performance counter (UPC):* UPCs gather performance counters and error counters from the other units.
- *Device control register (DCR) bus:* The DCR bus provides a direct interface from the processors to devices for configuration, control, and status information. Via the DCR bus, processors and the JTAG network are provided back-door access into the internal state of most units. This access proved invaluable to the development, debug, and verification of BLC. In all, there are approximately 750 DCRs on the BLC.

The following sections begin with a description of the ADE kernel and bootstrapper, and continue by detailing the low-level system programming interfaces (SPIs) provided for diagnostics and closely coded BG/L applications. The host-based development and support components of ADE are then discussed. Throughout these sections, we highlight the architectural features and programming concepts that all too often can be hidden under high-level, general-purpose (sometimes restrictive) application interfaces.

ADE bootstrapper and kernel

The BLC node software provided by ADE consists of a bootstrapper, kernel, system programming interface

[1]D. Hoenicke, M. A. Blumrich, D. Chen, A. Gara, M. E. Giampapa, P. Heidelberger, L.-K. Liu, M. Lu, V. Srinivasan, B. D. Steinmacher-Burow, T. Takken, R. B. Tremaine, A. R. Umamaheshwaran, P. Vranas, and T. J. C. Ward, "Blue Gene/L Global Collective and Barrier Networks," private communication.

libraries, and language support runtime libraries for C, C++, and Fortran. During early BLC logic development, ADE was flexibly used more as a toolkit than as a traditional operating system. From this toolkit, test-case designers constructed environments through compiletime and/or runtime configuration that best suited their needs. This flexibility was required for three reasons. First, different BLC modules were developed and integrated at different times. In fact, the very first BLC simulation environment consisted of a single PPC440 core model, without FPU, running against a simple SRAM memory model. Second, and perhaps most significantly, nonworking or unstable modules could be left in a quiesced state or, in most cases, held in reset. This allowed, for example, cycle-sim verification, described in [2], to proceed with embedded DRAM and L3 cache stubbed out of the BLC simulation model while awaiting a working cycle-sim embedded DRAM model. When hardware first arrived, the converse was true; BLC bring-up testing could proceed by running the kernel and test cases from embedded DRAM scratch space while we were debugging DDR controller initialization. Third, because event-sim logic simulations run $O(1)$ processor cycles per second of wall-clock time, productivity is improved by allowing test cases to focus on particular modules without having to wait through the full kernel initialization and configuration of other modules for each test run.

The flexibility this provided to BLC verification proved invaluable during the first few days of initial hardware bring-up. While the host-based hardware bring-up host console, described below, and service processor support interfaces were under development and debug, using ADE we were able to boot the hardware, make initial contact with all devices, and exercise the network interfaces, with the help of the IBM RiscWatch JTAG debugger. The first instructions run on the BLC application-specific integrated circuit (ASIC) chip were an ADE bootstrapper extension called *picoboot* that loaded exclusively into the uppermost 4 KB of SRAM, initialized both cores, verified SRAM functionality, and was used to debug the debugger itself. By the end of the third day of bring-up, the ADE kernel was pieced back together module-by-module as each device was shown to be functional, and a single-node fast Fourier transform (FFT) computational kernel was run from embedded DRAM scratch that produced the correct answer.

The majority of ADE test cases and applications fall into two categories: bootstrapper extensions or kernel extensions. Bootstrapper extensions are SRAM-based tests that are linked directly into the bootstrapper and executed following bootstrapper initialization. Examples of bootstrapper extensions include diagnostics for DDR, embedded DRAM, and L2 and L3 caches, all of which can perform content-destructive testing of 100% of memory. Other bootstrapper extensions have been created to act as monitors for DDR-based nonkernel diagnostics, including a program that generates random sequences of instructions and memory reference patterns and executes those instructions once in single-step mode and again in superscalar mode. Differences in results could indicate memory system or processor problems. The bootstrapper contains a very simple dual-threaded kernel capable of detecting and reporting any unexpected processor or device interrupts during BLC initialization. Because of the 16-KB size of SRAM, the bootstrapper cannot properly handle or recover from such interrupts; however, detection and reporting go a long way toward assisting the service processor or control system to diagnose any fatal early initialization problems.

Kernel extensions are linked directly into the kernel and are launched following kernel initialization by calling a defined entry point for each core in a separate privileged kernel thread within the single process (virtual address space) per node. By convention in kernel extensions, core 0, labeled the compute processor, is entered via a thread launch to the function `_CP_main()`; and core 1, labeled the I/O processor, is entered via a thread launch to `_IOP_main()`. In contrast to typical `pthread_create()` semantics, the arguments to each of these functions are identical to the familiar C language `main()` program entry point, including command-line arguments and environment variables. Kernel extensions have been developed using C, C++, and Fortran.

In ADE, we term this mode of operation *symmetric mode*. In symmetric mode, the ADE kernel provides a single-process multithreaded programming model that closely matches the BLC node architecture, in which the threads, like the cores, are full peers with equal access to all BLC hardware devices and their programming interfaces. This mode of operation, employed by the vast majority of diagnostics and verification codes, allows the creation of simultaneous coordinated attacks from both cores that more fully stress the hardware. Hardware exercisers, SPI collective communication routines, and applications use this mode to improve network performance by dividing injection and reception work between the cores. Two hardware facilities are provided to assist with interprocessor communication and control. Core-to-core interrupts provided by the BIC enable processors to deliver 32 standard and 32 critical interrupts, and even a machine check interrupt to their partner. ADE reserves five standard and two critical core-to-core interrupts for coordination, control, and error handling, and makes the rest available for test-case and application use via installable interrupt or signal handlers. Atomic operations between the processors are supported by the *lockbox*, which provides 256 low-latency

test-and-set semaphores and interprocessor barrier operations.

The ADE kernel, device drivers, SPI libraries, and language-support runtime libraries were designed and implemented to support multithreading with noncoherent L1 caches. Unlike normal symmetric multiprocessor (SMP) kernels, in which L1 cache management is predominantly used to gain a slight performance advantage, the coherence management in ADE is necessary for correctness. Like optimized SMP kernels, the ADE kernel organizes shared data structures on cache-line boundaries and avoids false sharing. For performance, larger data structures are aligned on L3 cache-line boundaries, reducing the total number of L3 cache operations and allowing a single L2 prefetch to access four L1 cache lines. Atomic access to data is guarded via the lockbox in conjunction with L1 cache invalidation before access and L1 flush after access. Critical code paths and device accesses are also semaphored via the lockbox. Symmetric mode applications and test cases use similar straightforward techniques. Software coherence management is not difficult; to date, the only bug encountered was a forgotten invalidate of a cache line that was dirtied when the Ethernet driver set the packet header checksum field to zero as part of validating the checksum. The asynchronous write-back of this dirty line overwrote the header of a later packet that reused the packet buffer via direct memory access (DMA).

ADE provides a choice of runtime libraries that support symmetric mode: SimLib and NewLib [8]. SimLib, scratch-written for ADE and primarily targeted for use in the simulation environments, requires zero start-up overhead and provides highly optimized small footprint implementations of needed language support functions. NewLib, used by only a few applications and hardware exercisers that require functionality of a more complete runtime, is particularly well suited to noncoherent multithreading because all static data has been organized into per-thread data structures.

The ADE kernel also provides a dual-process per-node programming model called *split mode*. In split mode, the kernel manages two processes, with each process assigned to one of the BLC cores. Split mode is entered by a kernel-managed `fork()` of the application data segment. To save physical memory and reduce translation lookaside buffer (TLB) pressure, a single kernel image is used, and through linkage conventions, large read-only application-constant areas are linked into the single code image shared by both processes. Because of the underlying symmetric mode support, split mode is a simple extension to more easily support the use of both cores for "dusty deck" compute-bound applications.

Configuration

ADE provides four default sets of compiletime build options and allows test-case and application designers to tailor any of these for their specific needs. The first three of these options, with certain restrictions, can flexibly move back and forth between the hardware and simulation environments. The first of these build options specifically targets the logic simulation environments. In this configuration, the kernel support for host console via the JTAG network is disabled. Simulation-only devices, such as the virtual universal asynchronous receiver transmitter (UART) serial output device and external network loop-back provided by the Very high-speed integrated circuit Hardware Description Language (VHDL) testbench, are enabled. Required workarounds for the differences between simulation and hardware are enabled. In addition, the ADE chip and kernel initialization is streamlined to save simulation time by taking advantage of the known BLC ASIC state at the start of simulation runs. For simulation runs, the bootstrapper and kernel have been deposited into memory, and the cache state is clean before reset is released.

The second ADE build option is used by test cases and some applications that require complete control over the BLC memory system. In this option, the default memory allocation code is disabled so that test cases can use the virtual memory management SPI of the kernel, described below, to allocate and configure memory in any or all of the possible modes.

High-level test cases, hardware exercisers, and applications predominantly use the third build option. In this option, the ADE kernel is responsible for the initialization, configuration, and monitoring of all BLC devices that have been enabled on the basis of the personalization. Memory management is provided through typical runtime library interfaces, subject to the configurations or restrictions imposed by the personalization described below. This build option also allows repeated runs of the same executable with various combinations of memory system configuration options.

The fourth build option for ADE completely transforms ADE into the BG/L ADE network interface emulator (BLNIE). With this option, the entire kernel and device interface for the hardware is replaced with an emulation layer that runs natively on IBM AIX* or Linux** host machines. There are two completely different uses for BLNIE builds of the ADE code base. First, a library has been provided that runs on the host and reaches into the BLC nodes via the JTAG interface to perform DCR accesses, processor state dumps, memory system peeks or pokes, and even back-door device accesses.

Specially written routines used much of the device driver code and register definitions from the host to debug the chip. This environment was invaluable in performing root-cause analysis of the few subtle memory system problems and even a processor erratum. The second use of the BLNIE build allows multinode, multicore applications and high-level test cases to be developed and executed on the host. This was used primarily after the logic design was completed, while we were waiting for hardware to arrive. In this environment, the system programming interfaces for the torus, collective, and global interrupt networks were implemented using shared memory for communication. Each BLC node was implemented as a host process, with a thread emulating each core of the BLC. Up to 16 emulated nodes, and depending on application workload and SMP host system performance, BLNIE performance was equal to BLC node performance. The value of BLNIE was proved when a simple recompile of many parallel test cases, including computational kernels from the Blue Matter [9] science team, ran on BG/L with no application-level code changes. The largest difference was that BLNIE emulated cores were coherent at L1, unlike BLC cores.

Personalization

Personalization of ADE is the last step before test cases and applications are loaded into the simulation environments or onto hardware partitions. Through personalization, ADE discovers the configuration of the parallel partition and how the test-case designer wants the various devices and memory system configured, and obtains application-level information, such as command-line arguments and environment variables. In both the bootstrapper and kernel, a memory area is reserved to hold the personalization information. These memory areas are written by a host program, called `svc_host`, that is included in ADE. Application designers supply arguments and flags to `svc_host` via shell scripts with each test case. Once personalization has been applied, `svc_host` computes and applies cyclic redundancy check (CRC) and checksum values of the bootstrapper, kernel, and personalization to ensure integrity of the load.

The bootstrapper SRAM personalization contains information needed for early BLC initialization by both the bootstrapper and kernel, including the following:

- Number and type of DDR memory modules installed.
- Field specifying DDR bit-steering parameters (currently unused).
- Enable/disable/configuration for L1, L2, L3, and scratch in all combinations.
- Enable/disable/configuration for torus, collective, global interrupts, and Ethernet networks.
- Whether this node is an I/O node or a compute node.
- Torus x, y, z dimensions and, for each dimension, whether torus or mesh.
- Node coordinates within the torus, or I/O node rank.
- Ethernet media access control (MAC) and Internet Protocol (IP) addresses.
- UPC initial programming configuration.
- Initial tracing masks to control error reporting and verbosity.
- Power control for idle units: torus on I/O nodes, Ethernet on compute nodes.
- Personality CRC and bootstrapper SRAM CRC.

The kernel personalization area contains application-specific information that is constant for all nodes in the partition, including the following:

- Command-line arguments.
- Environment variables.
- Miscellaneous SPI and application state variables.
- Kernel and test-case checksum.

Operation

From power-on reset, the service processor manages the start-up and configuration sequence of BLC via the JTAG network. In simulation, this is accomplished via Tcl/Tk scripts that deposit both the bootstrapper and kernel into memory. On hardware, the bootstrapper is written into the SRAM of each node and launched. All nodes receive the bootstrapper, typically done via JTAG broadcast. Unique personalization is then loaded into each node. Following processor and memory system initialization, the bootstrapper collaborates with the service processor to load the kernel through the JTAG mailbox messages. The bootstrapper supports three methods of kernel load: JTAG single-node load, JTAG broadcast, or single-node load followed by a broadcast via the internal high-speed network. Once the kernel load has been completed and verified, the bootstrapper flushes all cache state and transfers control to the kernel by simultaneously branching to the entry point of the kernel on each core. The kernel begins execution by reprogramming the core interrupt vectors to point to itself. During this short but vulnerable transfer of control, the simple interrupt handlers of the bootstrapper remain in effect to catch and report any severe node problems, such as those that might be encountered during initial manufacturing tests.

Following initial setup, the kernel, on the basis of compiletime configuration and personalization, enables and configures required devices, initializes the runtime

libraries and system programming interface libraries, creates kernel threads on each core, and invokes any C++ constructors. Finally, the kernel synchronizes processor cycle counters across the partition and then launches the application entry points simultaneously on all nodes. At that point, the ADE kernel becomes passive, awaiting device interrupts, reliability, availability, and serviceability (RAS) and error events, and kernel calls.

A feature of ADE, called *ping-pong reset*, ensures that the start-up sequence is deterministic, repeatable to the cycle, and recreatable in simulation. The bootstrapper, kernel, host console, and design of certain test cases all play a role in supporting this function. Ping-pong reset is initiated by invoking a kernel function on both cores that starts by flushing all levels of cache out to DDR and cleaning nondeterministic state from the cores and setting the soft-reset "cookie" in SRAM. This action has the side effect of causing the hardware state to match the initial condition used in simulation. Following completion of this step, one core toggles reset for the other core, then enters a spin loop. The reset core vectors off to the SRAM reset vector, sees the soft-reset cookie, and toggles reset for the other core. During this time, the host console suspends mailbox polling to avoid any SRAM interference that could introduce asynchrony in arbitration for SRAM access between the processors and the JTAG interface. In this way, the second and subsequent trips through ping-pong reset are fully repeatable to the cycle. Test cases that take advantage of this feature typically use iterative or random methods to explore a search space, saving the random seed used on each pass and invoking ping-pong reset between passes. Passes that showed unexpected behavior on the hardware could then be precisely rerun on other nodes to differentiate manufacturing defects in specific nodes, or rerun in simulation to catch logic errors. In several cases, a difference of even a single cycle could mask a problem. This feature of ADE was put to good use in performing root-cause analysis and finding and verifying workarounds of unexpected BLC behavior, and also recreation of processor errata. A minimal failing scenario could be found at the speed of hardware and then easily rerun in simulation for detailed analysis.

ADE system programming interfaces

The ADE system programming interfaces have been designed with several competing, often orthogonal, goals in mind. First, these interfaces had to be low-level and complete so that all aspects of the hardware could be specifically exercised. Second, they had to be sufficiently high-level to support multinode parallel test cases, benchmarks, and applications that stress the hardware to the limits of its capabilities. Third, they had to be general enough that combined test cases that stress multiple modules simultaneously could easily be developed and debugged. Fourth, to the extent possible, they had to support static compiletime setup and initialization to save simulation cycles, allowing the large regression suite to complete in reasonable time. Finally, through configuration and/or personalization, they had to support isolation of devices, so that, for example, during manufacturing test, faults in certain devices would not prevent us from testing other devices, providing more complete feedback to the quality control process. In the following paragraphs, each of the system programming interfaces is briefly introduced, and unusual features and functionality are highlighted. Details of specific examples using these SPI interfaces during verification, bring-up, and diagnostics have been described in [2, 6].

Most of the software in each of these SPI interfaces is devoted to RAS support and involves detailed error reporting and diagnosis, and, where possible, recovery. For each of the interfaces described here, the ADE kernel implements a device driver, consisting of initialization and configuration routines, and interrupt handlers to field normal or error condition device interrupts. RAS support makes up more than half of the entire kernel and runtime library footprint. ADE has been designed to remain responsive through nearly all likely hardware errors and configures each core to monitor the other core for machine checks or other fatal events.

Memory system management

The memory system management SPI enables two options for test cases and diagnostics: the choice of having complete control over the BLC memory system or leaving control to the kernel virtual memory manager (VMM). The choice can be made on the basis of the current configuration and personalization, through runtime library interfaces such as `malloc()` and `free()`, and SRAM allocation. The PPC440 processors implement a 36-bit physical address space. In BLC, the lower 32 bits of this address space are used to address memory and devices via memory-mapped I/O (MMIO). The upper four bits of the 36-bit address space are used as flags to control memory system configuration, including L2 inhibit, L3 inhibit, and bypass of the normal L2-to-PLB (processor local bus) interface, instead directing memory traffic via the on-chip peripheral bus (OPB). The PLB interface provides DMA support for the Gigabit Ethernet controller. The PPC440 address bus provides additional storage control attributes, expressed from *user attributes* in the processor TLBs onto the memory bus. In BLC, one of the user attributes is used to configure L2 prefetching for optimistic stream detection, or the default automatic stream detection, which requires confirmation to identify a stream. Another attribute is used as a flag to inhibit L3 prefetching from DDR.

The physical address space layout is pictured in **Figure 2**. The lower 2 GB of the address space maps DDR. Configurations of 256 MB, 512 MB, 1 GB, or 2 GB are supported. On BLC, one can partition the embedded DRAM between L3 cache and directly addressed memory, called *scratch space*, in multiples of 512 KB up to the full 4 MB, allowing software to create a high-speed shared scratch area that can contain directly addressed data or code. The BLC memory map uses two MMIO areas that allow flexible mapping into user space of the high-speed network interfaces and the lockbox. An additional MMIO area is used for the Ethernet. The 16-KB SRAM is located at the upper end of the 32-bit address space, which contains the PPC440 reset vector entry point. BLC boot is accomplished by writing initialization software directly into SRAM via the JTAG network and releasing reset from the processors. The *blind device* is an area in the physical address space where data written by the processors is immediately discarded by the memory system, and data read immediately returns garbage to the processors. This unusual device was designed for two reasons. The primary use of the blind device is as a cache-flush-assist device that speeds coherence management of the L1 caches of the processors. Software can use this area to quickly displace L1 cache contents by taking advantage of the L1 round-robin cache-line replacement policy. An upper bound of 1,024 cache-line touch or zero operations, one per cache line, can flush a much larger or irregular area of memory. Memory system diagnostic tests use this device as physical memory backing-store for locked L1 cache lines holding data and stack space, expanding their physical memory footprint beyond the 16-KB SRAM without perturbing the state of the memory system.

The low-level interfaces of the memory management SPI provide memory system diagnostics with a convenient error-checked way to perform their own TLB management with complete control over the L1, L2, and L3 caches, prefetching at L2 and L3, and enablement and size of scratch space. In all, this SPI provides control over 16 independently configurable memory system configuration options in all supported combinations and provides cache flush, invalidate, and reconfiguration functions for all levels of cache. Most of these options can be selected through kernel personalization, allowing an unmodified test case to execute under widely varying conditions. Randomly based test cases often use their seed to combine these options in different ways for better memory system coverage.

Lockbox

The ADE lockbox SPI provides spin-lock, try-lock, test-lock, force-lock, and intranode barrier functionality,

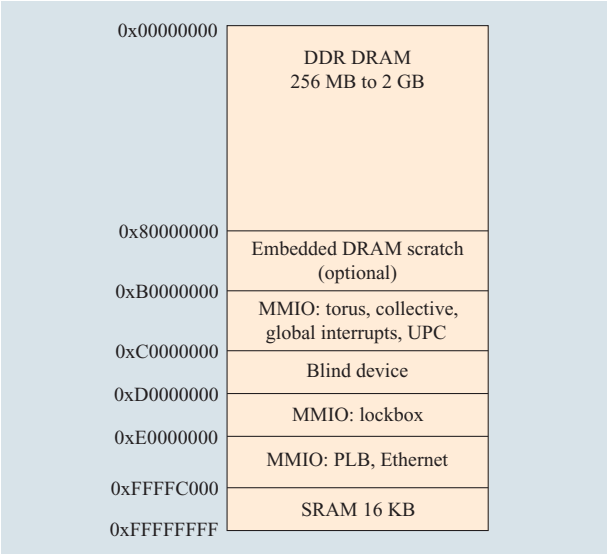

Figure 2

Blue Gene/L compute chip physical memory map.

available as low-latency inline functions or procedure calls. By using the VMM and the lockbox configuration DCR, locks can be assigned to the kernel, to specific cores, or to the application. The kernel uses locks to create critical code sections and, in coherence management, to protect data structures. The kernel also uses the lockbox to hold low-latency state variables to indicate that certain initializations have been completed. For example, in the bootstrapper and kernel start-up sequence, a try-lock operation is used to pick a core to perform memory system initializations, whether one or both cores have been released from reset. The second time through the initialization sequence, as would be the case during a host-initiated soft reset or the ping-pong reset operation discussed above, neither core wins the competition, thereby preserving the memory system state on reboot. Another interesting use of the lockbox, provided as a library interface, is for low-overhead, low-latency management of circular producer–consumer queues between the cores, in which a contiguous array of locks, one per queue slot, is used in place of the traditional memory-based head and tail pointers. However, the queue data must be coherence-managed, avoiding the latency to bounce the queue metadata off L3.

Torus SPI

The ADE torus SPI provides a packet-level interface based on the concept of active packets, whereby each arriving packet carries with it the address of a reception function, called an *actor*, that is triggered by packet arrival at the destination node. The SPI provides

two forms of active packets: buffered and unbuffered. Buffered packets are removed from the torus reception FIFOs (queues in which access takes place according to the first-in first-out rule) and placed in a reserved memory location before the actor is called with the address of that buffer. Buffered packets are used when the application requires random or unaligned access to the packet payload, as might be the case with a protocol packet. Because the reserved packet buffers are reused on a per-FIFO basis, buffered packets are most often received into and accessed from the processor L1 cache. Buffered packets are also used when a test case simply checks and then discards the packet contents. Unbuffered packets call the actor with the address of the reception FIFO containing the packet payload, providing zero copy reception, including the ability to receive payloads directly into processor registers. In addition to the location of the data, buffered and unbuffered actors are called with two parameters contained in the packet software header: a 32-bit untyped argument and an additional 10-bit untyped argument. As an example, a *put* actor might use the first argument as a destination address and the second argument as the number of bytes of payload to receive. An additional form of unbuffered active packet provides an extended software header that can have different meanings depending on the actor invoked. This extended header may, for example, contain additional control information required by SPI-supplied torus-class routing actors that must remove and reinject packets at corner turns to avoid network deadlocks for plane filling or broadcasts on the torus.

The torus SPI provides a full set of header creation and manipulation routines for hardware and software headers. Depending on the options desired, torus header creation could require tens of cycles to well over 100 processor cycles, for example when calculating hint bits to apply to the packet based on the relative coordinates of the destination node. The SPI allows headers to be created during the setup phase of the application or created on the fly based on templates. To speed simulation, headers or complete packet images can be created at compiletime.

The SPI provides blocking and nonblocking packet send interfaces that choose the injection FIFO on the basis of space available and packet destination and a lower-level injection interface where the test case or application selects the injection FIFO. Similarly, the reception interface supports blocking or nonblocking polling, with round-robin fairness among the reception FIFOs. Actor functions are launched via the polling functions in the context of the thread performing the polling. Library routines built up from the SPI, described in [6], support point-to-point messaging, row multicast, and plane- and subcube-filling communication algorithms.

Collective network SPI

In many respects, the collective network SPI is simpler than the torus SPI. The collective network supports fixed-size packets of 256 bytes and always delivers packets in order. The collective network requires a simple hardware control header that selects the class route to use, whether the packet is point-to-point or collective and interrupting or noninterrupting. A software header is optional and generally used only on point-to-point packets.

The collective network SPI implements blocking and nonblocking send and receive routines and lower-level raw injection and reception routines. Built atop this base, the SPI implements collective broadcast and reduce function. Because of the multithreaded ADE programming model, peak collective network performance can be more easily achieved in complex collective operations by dividing the workload between the two cores, using one processor to handle the injection side while the other processor handles the reception side.

Capture-unit SPI

The capture-unit SPI is implemented both in the kernel and in the bootstrapper. For multiple midplane partitions, the bootstrapper enables the capture units to send training patterns during early BLC chip initialization so that the BG/L link (BLL) chips that interconnect the multiple midplanes can be configured and trained by the host console. On the basis of JTAG mailbox messages from the host console, the bootstrapper can complete and error-check the training sequence in preparation for collective network broadcast of the kernel. Alternatively, the kernel device driver can complete the training sequence.

The capture-unit SPI also contains library calls to enable or disable automatic error injection in the capture units. This function is used by stressful diagnostics to verify recovery hardware in the presence of more frequent errors than would normally be encountered.

Global interrupt SPI

The global interrupt SPI supports the creation and management of partition-wide barriers and notifications or alerts. Early in its start-up, the kernel uses the global interrupt SPI for coordination of capture-unit training and global clock synchronization. By default, two barriers are created: one that includes compute and I/O nodes and another that includes only compute nodes. Notifications are configured to deliver partition-wide interrupts that signal error conditions, or state change, with optional user-installable interrupt or signal handlers.

Ethernet SPI

The ADE kernel Ethernet driver supports industry-standard protocols including User Datagram Protocol (UDP), Address Resolution Protocol (ARP), and a subset of Internet Control Message Protocol (ICMP), including ping. For normal packet transmission or reception, the Ethernet programming interface provides a choice of interrupt mode or polling mode, independently for the send side and the receive side. For Ethernet device error interrupts, the Ethernet driver provides handlers that attempt recovery for nonfatal errors or report any fatal errors, such as an unplugged Ethernet cable, as RAS events to the host console. If interrupts are desired, they can be directed, or funneled, to either core to be handled by installable callback routines that operate much like signal handlers. For diagnostics use, the Ethernet driver provides a choice of external PHY (physical layer of the Ethernet) loopback on I/O nodes or internal Ethernet media access controller (EMAC) loopback on all nodes. This has allowed greater coverage in test cases by enabling Ethernet traffic on compute nodes during memory system stress tests. The Ethernet SPI also provides an interface to power down the entire Ethernet subsystem of the BLC ASIC on compute nodes.

Universal performance counters SPI

The UPC provides performance and error counters for most BLC devices and, from a choice of hundreds, allows up to 48 counters to be active at a time. The SPI provides the ability to enable and disable counters, program the counters by event name, and report and clear the count; it also provides field interrupts based on count thresholds. The SPI sets up a default UPC configuration that enables all UPC error counters to generate RAS events that are captured and reported to the service processor. For speeding simulation runs, or for multiple runs of the same test-case binary using different sets of counters, programming of the UPC can be done on the host using the ADE `svc_host` utility and is provided to the kernel via the personalization at load time as a prechecked and preconfigured list of DCR values to be programmed into the UPC.

Trace, logging, and debug

The tracing and logging function in the ADE kernel is fully controllable at compiletime or runtime. If this support is enabled at compiletime, each trace point or trace category is controlled by an array of bit masks that can be set by personalization at loadtime or set at runtime. This provides the ability to completely silence kernel tracing and logging, which proved useful when moving between the simulation and hardware environments, since these use different external interfaces to gather this information. Typically, a kernel I/O module must be recompiled when switching between these environments, but by avoiding this difference, the same binary code could run in both places, allowing perfect reproducibility of a particular BLC issue that could happen only during a single-cycle window. For extremely noninvasive tracing, ADE provides an interface to three of the Special Purposed Registers General (SPRG) of the PPC440 that can be accessed in a single instruction to record status or history information.

ADE supports two forms of code profiling—periodic and histogram—that enable analysis of application performance by identifying code and algorithmic hot spots. Periodic profiling captures the code instruction pointer at regular and repeated intervals, ranging from a minimum of 2 μs to a maximum of several seconds. Histogram profiling allows the user to select a code range and granularity of interest and count the number of sample periods during which that code granule was executing.

In addition to RiscWatch debugger support discussed above, the ADE kernel provides a full traceback of the call history upon any application fault or addressing error. Code and data debug support uses the PPC440 built-in debug facilities that allow hardware-managed breakpoints to be enabled for up to two specific code addresses, or inclusive or exclusive code ranges. Data watch-points are supported in a similar fashion.

Compilers and development tools

ADE provides a cross-development platform, supported on Linux and AIX, consisting of the GNU compiler collection (GCC) and the full suite of binutils.[2] Early in the project, support for the embedded PowerPC cores in GCC and the binutils was limited to the PowerPC 403*, a much simpler embedded core. Much of the work involved was to enable the PPC440 and double-FPU instruction set, and to tune the GCC machine description to understand the PPC440 superscalar architecture, including the complex integer I-pipe, the simple integer and system instruction J-pipe, the load and store L-pipe, and the floating-point F-pipe. More recently, with the release of the GNU Compiler Collection (GCC) 3.4, the Open Source community has graciously updated GCC and the binutils to better understand the PPC440, causing much of this early work to be obsolesced. Migration to this newer release is underway. However, support for the double FPU [4]—and the application binary interface (ABI) changes it requires for quadword (16-byte) alignment—is still necessary. In addition, a newly discovered PPC440 erratum has required a simple workaround in the compiler code generation back end.

[2] BINary UTILities, which support the GNU compilers by providing programs that manipulate binary (machine-readable but not human-readable) object code and executable files.

Host console

The BG/L bring-up host console provides an integrated host environment to run ADE programs. It supplies a centrally maintained machine configuration file that defines separate physical partitions with compute and I/O nodes. A user then specifies the partition name and a list of nodes on which to run programs. The node ranges are specified in terms of their x, y, z coordinates on the three-dimensional torus.

The host console and a collection of host utilities provide the following functionality:

- Initializing all BG/L nodes within a partition.
- Setting up the global barrier network and training link chips in the BG/L midplane when necessary.
- Personalizing ADE application programs with x, y, z coordinates for each node, along with user-supplied arguments and environment variables.
- Loading and running ADE programs.
- Polling each running node for debug and status print outputs.
- Logging error messages in a central RAS log when hardware problems are found.

To the extent possible, the host console provides this function in a manner similar to that used in the simulation environments. This better supports the ability to move test cases and diagnostics back and forth between hardware and simulation.

Conclusion

In this paper, the design and use of the Blue Gene/L advanced diagnostics environment have been presented, with a focus on key architectural programming features of BG/L to assist high-performance application developers to more fully exploit the capabilities of the machine. ADE has been used to create many hundreds of test cases, ranging from short, simple tests that verify correct operation of as little as a single DCR in the design, to large, massively parallel hardware exercisers that stress many thousands of nodes, probing for failures and incorrect operation in ways that would generally not be possible for traditional operating systems and applications. Subsets of these tests have been consolidated into the *regression suite*, run continually during logic development for verification; the *manufacturing test suite*, used for acceptance and burn-in of new hardware; and the *diagnostics suite*, used for ongoing BG/L machine maintenance and test. Experience is being gained on a daily basis as the BG/L machine is scaled. The diagnostics environment, tests, and tools will continue to evolve to meet the RAS challenges of the BG/L machine.

Acknowledgment

The authors gratefully acknowledge the hard work, test-case development, support, suggestions, and advice for improving ADE provided by the following individuals: Daniel Beece, Lurng-Kuo Liu, Narasimha R. Adiga, Balaji Gopalsamy, Arun R. Umamaheshwaran, Krishna M. Desai, Blake G. Fitch, Aleksandr Raysubskiy, T. J. C. Ward, James C. Sexton, George Chiu, and Paul Coteus.

The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Lawrence Livermore National Laboratory Subcontract No. B517552.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds in the United States, other countries, or both.

References

1. A. Gara, M. A. Blumrich, D. Chen, G. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L System Architecture," *IBM J. Res. & Dev.* **49**, No. 2/3, 195–212 (2005, this issue).
2. M. E. Wazlowski, N. R. Adiga, D. K. Beece, R. Bellofatto, M. A. Blumrich, D. Chen, M. B. Dombrowa, A. Gara, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, B. J. Nathanson, M. Ohmacht, R. Sharrar, S. Singh, B. D. Steinmacher-Burow, R. B. Tremaine, M. Tsao, A. R. Umamaheshwaran, and P. Vranas, "Verification Strategy for the Blue Gene/L Chip," *IBM J. Res. & Dev.* **49**, No. 2/3, 303–318 (2005, this issue).
3. IBM Corporation, IBM Document SA14–2523: IBM PPC440 Core User's Manual; see *http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/242E6A5364DF2EDE87256AE9005BD327*.
4. K. Dockser, "'Honey, I Shrunk the Supercomputer'—The PowerPC 440 FPU Brings Supercomputing to IBM's Blue Logic Library," *IBM MicroNews* **7**, No. 4, 29–31 (November 2001).
5. Book-E Enhanced PowerPC Architecture; see *http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF778525699600682CC7*.
6. N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue Gene/L Torus Interconnection Network," *IBM J. Res. & Dev.* **49**, No. 2/3, 265–276 (2005, this issue).
7. R. A. Haring, R. Bellofatto, A. A. Bright, P. G. Crumley, M. B. Dombrowa, S. M. Douskey, M. R. Ellavsky, B. Gopalsamy, D. Hoenicke, T. A. Liebsch, J. A. Marcella, and M. Ohmacht, "Blue Gene/L Compute Chip: Control, Test, and Bring-Up Infrastructure," *IBM J. Res. & Dev.* **49**, No. 2/3, 289–301 (2005, this issue).
8. NewLib; see *http://sources.redhat.com/newlib/*.
9. R. S. Germain, Y. Zhestkov, M. Eleftheriou, A. Rayshubskiy, F. Suits, T. J. C. Ward, and B. G. Fitch, "Early Performance Data on the Blue Matter Molecular Simulation Framework," *IBM J. Res. & Dev.* **49**, No. 2/3, 447–455 (2005, this issue).

Received July 6, 2004; accepted for publication October 18, 2004; Internet publication April 12, 2005

Mark E. Giampapa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (giampapa@us.ibm.com)*. Mr. Giampapa is a Senior Engineer in the Exploratory Server Systems Department. He received a B.A. degree in computer science from Columbia University. He joined the IBM Research Division in 1984 to work in the areas of parallel and distributed processing, and has focused his research on distributed memory and shared memory parallel architectures and operating systems. Mr. Giampapa has received three IBM Outstanding Technical Achievement Awards for his work in distributed processing, simulation, and parallel operating systems. He holds 15 patents, with several more pending, and has published ten papers.

Ralph Bellofatto *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ralphbel@us.ibm.com)*. Mr. Bellofatto is a Senior Software Engineer. He has been responsible for various aspects of hardware system verification and control system programming on the Blue Gene/L project. He received B.S. and M.S. degrees from Ithaca College in 1979 and 1980, respectively. He has worked as a software engineer in a variety of industries. Mr. Bellofatto's interests include computer architecture, performance analysis and tuning, network architecture, ASIC design, and systems architecture and design. He is currently working on the control system for Blue Gene/L.

Matthias A. Blumrich *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (blumrich@us.ibm.com)*. Dr. Blumrich is a Research Staff Member in the Server Technology Department. He received a B.E.E. degree from the State University of New York at Stony Brook in 1986, and M.A. and Ph.D. degrees in computer science from Princeton University in 1991 and 1996, respectively. In 1998 he joined the IBM Research Division, where he has worked on scalable networking for servers and the Blue Gene supercomputing project. Dr. Blumrich is an author or coauthor of two patents and 12 technical papers.

Dong Chen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (chendong@us.ibm.com)*. Dr. Chen is a Research Staff Member in the Exploratory Server Systems Department. He received his B.S. degree in physics from Peking University in 1990, and M.A., M.Phil., and Ph.D. degrees in theoretical physics from Columbia University in 1991, 1992, and 1996, respectively. He continued as a postdoctoral researcher at the Massachusetts Institute of Technology from 1996 to 1998. In 1999 he joined the IBM Server Group, where he worked on optimizing applications for IBM RS/6000* SP systems. In 2000 he transferred to the IBM Thomas J. Watson Research Center, where he has been working on many areas of the Blue Gene/L supercomputer and collaborating on the QCDOC project. Dr. Chen is an author or coauthor of more than 30 technical journal papers.

Marc Boris Dombrowa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (dombrowa@us.ibm.com)*. Mr. Dombrowa received his Dipl.-Ing. degree in electrical engineering from the University of Hannover, Germany, in 1997. He was a very large scale integration (VLSI) designer at the IBM VLSI Laboratory in Boeblingen, Germany, from 1997 to 1998, performing memory design verification and synthesis on S/390* Enterprise memory systems. From 1998 to 2000 he was assigned to the S/390 Server Division at the IBM Poughkeepsie facility to perform custom circuit design. He moved to Blue Gene/L cellular systems chip development in 2001 and has been responsible for the high-level design, synthesis, timing, and verification of the test interface of the Blue Gene/L compute chip as well as design-for-testability transformation for the entire chip, clock-tree verification, and simulation setup for instruction program load for the chip verification teams. Mr. Dombrowa received an IBM Outstanding Achievement Award in 1998 for his S/390 contributions. He is co-inventor of one patent. His research interests include computer architecture, design for test, system bring-up, diagnostics, and ASIC design. Mr. Dombrowa is currently working on the manufacturing diagnostic software as well as the system-level rack diagnostic test suite and bring-up for the Blue Gene/L cluster.

Alan Gara *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (alangara@us.ibm.com)*. Dr. Gara is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the University of Wisconsin at Madison in 1986. In 1998 Dr. Gara received the Gordon Bell Award for the QCDSP supercomputer in the most cost-effective category. He is the chief architect of the Blue Gene/L supercomputer. Dr. Gara also led the design and verification of the Blue Gene/L compute ASIC as well as the bring-up of the Blue Gene/L prototype system.

Ruud A. Haring *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ruud@us.ibm.com)*. Dr. Haring is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received B.S., M.S., and Ph.D. degrees in physics from Leyden University, the Netherlands, in 1977, 1979, and 1984, respectively. Upon joining IBM in 1984, he initially studied surface science aspects of plasma processing. Beginning in 1992, he became involved in electronic circuit design on both microprocessors and application-specific integrated circuits (ASICs). He is currently responsible for the synthesis, physical design, and test aspects of the Blue Gene chip designs. Dr. Haring has received an IBM Outstanding Technical Achievement Award for contributions to the z900 mainframe, and he holds several patents. His research interests include circuit design and optimization, design for testability, and ASIC design. Dr. Haring is a Senior Member of the IEEE.

Philip Heidelberger *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (philiph@us.ibm.com)*. Dr. Heidelberger received a B.A. degree in mathematics from Oberlin College in 1974 and a Ph.D. degree in operations research from Stanford University in 1978. He has been a Research Staff Member at the IBM Thomas J.

Watson Research Center since 1978. His research interests include modeling and analysis of computer performance, probabilistic aspects of discrete event simulations, parallel simulation, and parallel computer architectures. He has authored more than 100 papers in these areas. Dr. Heidelberger has served as Editor-in-Chief of the *ACM Transactions on Modeling and Computer Simulation*. He was the general chairman of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS) Performance 2001 Conference, the program cochairman of the ACM SIGMETRICS Performance 1992 Conference, and the program chairman of the 1989 Winter Simulation Conference. Dr. Heidelberger is currently the vice president of ACM SIGMETRICS, and is a Fellow of the ACM and the IEEE.

Dirk Hoenicke *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (hoenicke@us.ibm.com)*. Mr. Hoenicke received a Dipl. Inform. (M.S.) degree in computer science from the University of Tuebingen, Germany, in 1998. Since then, Mr. Hoenicke has worked on a wide range of aspects of two prevalent processor architectures: ESA/390 and PowerPC. He is currently a member of the Cellular Systems Chip Development Group, where he focuses on the architecture, design, verification, and implementation of the Blue Gene system-on-a-chip (SoC) supercomputer family. In particular, he was responsible for the architecture, design, and verification effort of the collective network and defined and implemented many other parts of the BG/L ASIC. His areas of expertise include high-performance computer systems and advanced memory and network architectures, as well as power-, area-, and complexity-efficient logic designs.

Gerard V. Kopcsay *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (kopcsay@us.ibm.com)*. Mr. Kopcsay is a Research Staff Member. He received a B.E. degree in electrical engineering from Manhattan College in 1969 and an M.S. degree in electrical engineering from the Polytechnic Institute of Brooklyn in 1974. From 1969 to 1978 he was with the AIL Division of the Eaton Corporation, where he worked on the design and development of low-noise microwave receivers. He joined the IBM Thomas J. Watson Research Center in 1978. Mr. Kopcsay has worked on the design, analysis, and measurement of interconnection technologies used in computer packages at IBM. His research interests include the measurement and simulation of multi-Gb/s interconnects, high-performance computer design, and applications of short-pulse phenomena. He is currently working on the design and implementation of the Blue Gene/L supercomputer. Mr. Kopcsay is a member of the American Physical Society.

Ben J. Nathanson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (bjnath@us.ibm.com)*. Mr. Nathanson joined the IBM Research Division in 1985 and has worked on the parallel computers RP3, Vulcan, SP1, SP2, and Blue Gene/L. He received IBM Outstanding Technical Achievement Awards for hardware contributions to SP1 and SP2 and Research Division Awards for RP3 bring-up and verification work on memory compression hardware. Mr. Nathanson holds M.S. and B.S. degrees in electrical engineering from Columbia University and is a member of Tau Beta Pi and Eta Kappa Nu. His current focus is hardware verification.

Burkhard D. Steinmacher-Burow *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (steinmac@us.ibm.com)*. Dr. Steinmacher-Burow is a Research Staff Member in the Exploratory Server Systems Department. He received a B.S. degree in physics from the University of Waterloo in 1988, and M.S. and Ph.D. degrees from the University of Toronto in 1990 and 1994, respectively. He subsequently joined the Universitaet Hamburg and then the Deutsches Elektronen-Synchrotron to work in experimental particle physics. In 2001, he joined the IBM Thomas J. Watson Research Center and has since worked in many hardware and software areas of the Blue Gene research program. Dr. Steinmacher-Burow is an author or coauthor of more than 80 technical papers.

Martin Ohmacht *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mohmacht@us.ibm.com)*. Dr. Ohmacht received his Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from the University of Hannover, Germany, in 1994 and 2001, respectively. He joined the IBM Research Division in 2001 and has worked on memory subsystem architecture and implementation for the Blue Gene project. His research interests include computer architecture, design and verification of multiprocessor systems, and compiler optimizations.

Valentina Salapura *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (salapura@us.ibm.com)*. Dr. Salapura is a Research Staff Member with the IBM Thomas J. Watson Research Center, where she has contributed to the architecture and implementation of three generations of Blue Gene systems (BG/C, BG/L, and BG/P), focusing on multiprocessor interconnect and synchronization and multithreaded architecture design and evaluation. She received a Ph.D. degree from the Vienna University of Technology in 1996. Before joining IBM in 2000, she was a faculty member with the Computer Engineering Department at the Vienna University of Technology. In addition to her work on high-performance systems, she has been a driving force in the design and evaluation of the SanLight network multiprocessor architecture. Dr. Salapura is the author of more than 50 papers on design methodology, configurable architectures, network processors, and high-performance computer systems; she holds one patent and has 19 patents pending.

Pavlos Vranas *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (vranasp@us.ibm.com)*. Dr. Vranas is a Research Staff Member in the Deep Computing Systems Department at the IBM Thomas J. Watson Research Center. He received his B.S. degree in physics from the University of Athens in 1985, and his M.S. and Ph.D. degrees in theoretical physics from the University of

California at Davis, in 1987 and 1990, respectively. He continued research in theoretical physics as a postdoctoral researcher at the Supercomputer Computations Research Institute, Florida State University (1990–1994), at Columbia University (1994–1998), and at the University of Illinois at Urbana–Champaign (1998–2000). In 2000 he joined IBM at the Thomas J. Watson Research Center, where he has worked on the architecture, design, verification, and bring-up of the Blue Gene/L supercomputer and is continuing his research in theoretical physics. Dr. Vranas is an author or coauthor of 59 papers in supercomputing and theoretical physics.